L.I.S.T. Group

LONG ISLAND
SINCLAIR TIMEX
GROUP

TO:

10 Idle Day Drive
Centerport, N.Y. 11721

Dear Member:

We have a core group now and should schedule a meeting to discuss dues, goals etc. Please call me at the number listed below, and we'll try to arrange a time and place. For openers, lets consider April 1st (Sunday) at 3:00PM, my place.

As for dues, I think $12.00/year will get us started and cover initial costs.

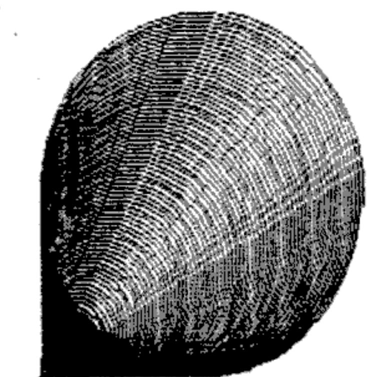
We'll need volunteers to take on the various functions, e.g., maintaining current member lists, mail drop, meeting arrangements, newsletter, so please think about what you'd like to do.

Please send us any ideas, questions or articles you have written, for inclusion in the newsletter. Also, drop us a line on your system configuration.

Very truly yours,

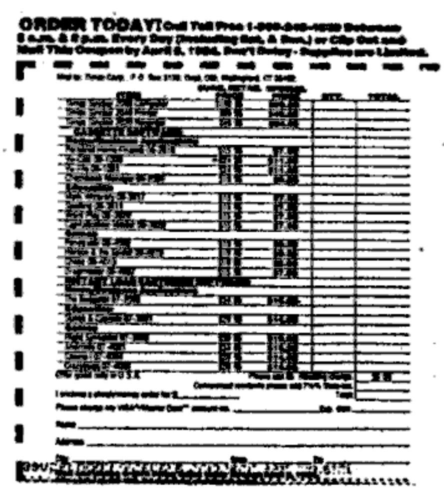
Paul J. Donnelly

W - 212-572-7603
H - 516-261-6934



LONG ISLAND SINCLAIR TIMEX GROUP

NEWSDAY, SUNDAY, MARCH [illegible], 1984



O
P1

LEDOUT VERSION 1.2

COPYRIGHT AUG 1983 [illegible]

```
  1 REM ....................
KEYBO  COPY  ... ... P
RINT ...  DO NOT USE THIS CODE
BCUS ?  SEE RECORD FIG. 2(b)
?RND)E ... RND ... 
 10 REM   LEDOUT VERSION 1.2
 11 REM        AUG 30 1983
100 REM  INITIALIZE
101 POKE 16514,0
102 POKE 16538,0
103 POKE 16539,0
105 POKE 16540,255
110 LET MENU=1000
111 REM MENU IS FOR MAIN
        PROGRAM
121 LET LEDOUT=16542
122 LET SETUP=16535
124 GOTO MENU
125 REM LEDOUT
127 PRINT ,,"INPUT DECIMAL VALU
E"
130 INPUT D
131 PRINT D
134 REM  NOW CONVERT D TO 3 SEP
ARATE DIGITS
135 LET H=INT (D/100)
137 LET T=INT ((D-H*100)/10)
139 LET U=INT (D-(H*100)-(T*10)
140 REM NOW CHANGE THAT TO OUR
        SPECIAL CODE
142 LET H=H+64
144 LET T=T+32
146 LET U=U+16
148 REM COULD HAVE BEEN DONE IN
        135 THRU 139
150 POKE SETUP,H
152 POKE SETUP+1,T
154 POKE SETUP+2,U
158 REM  CALL LEDOUT
159 FAST
160 LET X=USR LEDOUT
162 SLOW
165 RETURN
```

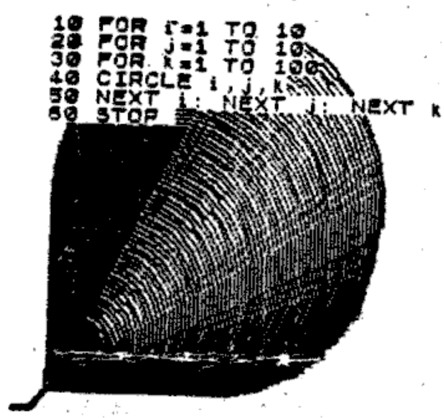
Fig. 3(a)
LEDOUT

```
200 REM AD OUT
204 REM CHOOSE CHANNEL 0
205 POKE 16514,0
210 FAST
215 LET L=USR 16602
217 REM  R IS SOME SCALING
         FACTOR
218 LET X=1
220 LET D=X*PEEK 16515
225 SLOW
230 GOSUB 130
299 GOTO MENU
300 PRINT "    ";
```

Fig. 3(b)
BASIC LEDOUT
FOR A/D CONVERTER

L.I.S.T.

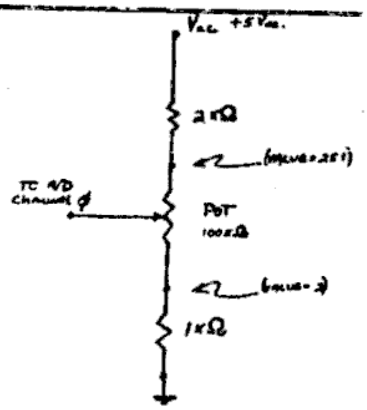
JUST FOOLING AROUND With 3D



What's wrong with this program?

```
  1 REM ....................
      THIS IS A NUMB AREA
  4 FAST
 10 REM  BASIC LEDOUT
 11 LET H=0
 12 LET T=0
 13 LET U=0
 15 LET OUTPUT=16646
 16 LET BUF=16633
 20 DIM V(4)
100 REM MAIN
102 SLOW
105 PRINT "INPUT DECIMAL VALUE"
110 INPUT D
112 PRINT D
115 LET H=64+INT (D/100)
120 LET T=32+INT ((D-(H-64)*100
)/10)
125 LET U=16+INT (D-(H-64)*100-
(T-32)*10)
130 LET V(1)=H
135 LET V(2)=T
140 LET V(3)=U
145 LET V(4)=0
150 FOR J=1 TO 50
155 FOR I=1 TO 3
160 POKE BUF,V(I)
162 FAST
165 LET X=USR OUTPUT
170 NEXT I
180 NEXT J
182 POKE BUF,V(4)
185 LET I=USR OUTPUT
190 PRINT "AGAIN?"
192 INPUT A$
193 IF A$="Y" THEN GOTO 105
194 IF A$="D" THEN GOTO 200
195 STOP
199 REM BLEDOUT
200 POKE 16514,0
205 FAST
210 LET L=USR 16602
220 LET U=PEEK 16515
222 SLOW
225 GOTO 115
230 STOP
```

Fig. 4
Pure BASIC LEDOUT
(w/ A-to-D)




```
5999 REM ADTEST
6005 FAST
6010 FOR I=1 TO 10
6020 POKE 16514,0
6030 LET L=USR 16602
6040 LPRINT "VALUE=";PEEK 16515
6050 NEXT I
6060 REM ...
```

TYPICAL VALUE RANGE

```
VALUE=251
VALUE=251
VALUE=241
VALUE=189
VALUE=160
VALUE=98
VALUE=71
VALUE=26
VALUE=2
VALUE=2
```

HARDWARE/REPAIR TIPS

If your ZX/TS keyboard starts acting up, it may be because of poor contact between the keyboard's flat ribbon conductor and the mainframe. I lost the bottom 5 keys on the right hand side recently for no apparent reason. A quick look at the ZX81 schematic shows these keys (B,N,M,.,Break) to be controlled by the [illegible] line on the 8 connector block. Apparently, the very thin metal film on the plastic carrier can be scraped off during assembly.

Pull the ribbons out of their plugs and hold the ends up to a light. If the metal film was scratched off, you'll see daylight through the traces. Repair is simple, just take a sharp pair of scissors and trim exactly 1/8" from the entire bottom edge of the flat cable (don't forget the notch).

Diagnosing the problem is just as easy - If you loose columns (e.g., [illegible] and [illegible]) you've lost contact with one of the [illegible] lines (in this case [illegible]) on the 5 pin connector. If you are lost (as to my case), you'll loose 5 keys, so suspect the eight pin lines.

Sample output from VU-3D

MAG=001.45 ROT 072,245 Z=+00250

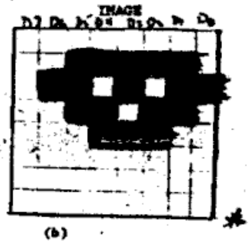
WHAT A CHARACTER

The Game Changer Interface (reviewed elsewhere) enables you to upload ATARI VCS (2600) cartridge contents into your ZX/TS, SAVE them on tape and play them, using the ZX/TS RAM. Since the copy of the game is in RAM, you can also change any part you wish, to suit your own personal style. The ATARI uses a 6507 (processor) significant changes require programming in 6502 machine code.

However, an understanding of 6502 machine code (MC) is not really necessary for you to change some aspects of your recorded ATARI games. Specifically, any character or icon (a symbol representing you, an object, enemy etc.) can be changed to another shape, just by POKEing new values into its data array.

In order to be represented on the screen, an object must have "bit image" stored somewhere in memory. The game software sends these bits to the hardware which in turn produces the picture you see on the screen. Here's a simple example of a face using an 8 X 8 bit image (that is, 8 bits wide by 8 bits high):

Binary Code	CODE(Hex)
00000000	00
00111110	3E
01101011	6B
00110110	36
00011100	1C
00000000	00
00000000	00
00000000	00
(a)	(c)



(b) *

Fig. 1

Figure 1(a) gives the binary representation of our "face". Here ones (1's) will represent "turned on" bits and will be dark, while '0's are turned off or blank (the background color will show through). In Fig. 1 c), since this image is exactly 8 bits wide, we've shown the hexadecimal numbers representing one bit byte. This face can be stored in 8 consecutive memory locations.

If we were to pull up the ATARI game code on the screen a byte at a time and represent it as a series of black and white squares we could "see" the actual character images used in the game. Fig. 2 gives a partial program listing that will do just that.

Before you GOTO 7000 however, you may have to do a little detective work. A typical ATARI cartridge contains a 4K ROM which you will already have downloaded to the 4K of RAM starting at 16514. The 6502 interrupts (where the program restarts after something important or bad happens) point to the very high memory locations and addresses pointing to the actual program code will be stored there. You can use the disassembler or monitor to check the top sections of RAM (FFFA through FFFF) for these pointers. You can be reasonably sure the data tables for the characters are not in the immediate areas pointed to by those three sets of 2 bytes and should look elsewhere. Places to look are; 1) boundaries between pages (F000, F300 etc. - a "page" is 256 bytes) 2) the beginning of the program 3) addresses which occur frequently.

For ATARI BASIC we used the disassembler to spot check some of these areas and found absolute garbage at the beginning of the program. A good hard look at the 1 REM (see fig 3) also shows the experienced programmer something which just about has to be nonsense. The reoccurrence of the same code number several times in succession is very unlikely in an MC program. These clues led us to choose 16514 as a starting address. After offset, this address corresponds to F000(H) in the ATARI.

After GOing to 7000 we obtained the listing in Fig. 4. A pattern of dots is clearly emerging, but hard to identify as letters or figures. Connecting the dots (or using the inverse space instead of an asterisk) can help make the patterns more intelligible. We've done that for the "GOTO" command of ATARI BASIC. You should be able to pick out some other upside-down commands like Print, and Else. The game software and hardware automatically turn these right side up.

How does the routine work? First we dimension some new variables and pick a starting address. These are:

- S = Starting address (decimal only)
- D(I) = The binary i th Digit of a Byte
- A$ = A string which will print either filled-in or MT squares
- P = The decimal value of the byte at location S
- N = A temporary "part" of the P variable as it is reduced to its D or binary digits.

Next we PEEK the data stored in the currently chosen memory location. The subroutine at 8000 repeatedly divides the decimal number by 2, saving the remainder as D(J) to convert the decimal number to its binary (actually binary still stored as decimal) bit equivalents.

Returning to 7025ff from the subroutine we convert each "character" in the string A$ to its bit pattern: it remains a blank if D(J)≠1 (that is, if its 0) and becomes an asterisk if D(J)= 1.

Finally we print the memory location, decimal value and our new blocked out string of "bits". Don't forget that the actual addresses where this data table reside are some 32000 (decimal) memory locations higher in the ATARI. Also, the conventions used by this programmer may not be used in other games. In ATARI BASIC, in fact, two more, different, techniques are used for addressing the numbers (for MATH) and the display area titles.

You should be able to add these lines to the "change" program with no problems on 4K or less games. You could even just pick an address at random and observe the screen for intelligible patterns. For 8K games create a free standing version of the program addition and try to PEEK the (with the actual cartridge installed) memory locations above 18000. Remember, the bank switching may get in the way.

Some hints as what to change in ATARI BASIC:

1) The symbol table for the display areas (STATUS, PROGRAM etc.) Abbreviating these would free up RAM for your own 6502 MC routines.
2) Change the "Block" graphics character to more interesting picture (e.g., a smiley face).
3) Change the alpha or numeric characters to provide a more interesting display. The letters, for example, can be converted to those of another language.
4) Your own ideas.

```
7000>DIM D(8)
7003 DIM A$(8)
7005 PRINT " INPUT STARTING ADDR
ESS..."
7007 INPUT S
7010 FOR I=S TO S+2000
7015 LET P=PEEK I
7020 GOSUB 8000
7025 FOR J=1 TO 8
7030 LET A$(J)=" "
7040 IF D(J)=1 THEN LET A$(J)="*"
7045 NEXT J
7050 PRINT I;"   ";PEEK I,A$( TO 8)
7060 NEXT I
7900 STOP
8000 FOR J=8 TO 1 STEP (-1)
8010 LET N=INT (P/2)
8020 LET D(J)=INT (P-2*INT (P/2+.05))
8030 LET P=N
8040 NEXT J
8050 RETURN
9000 STOP
```

Fig. 2

Fig. 3

* This is Technically termed a "Shape Table"!

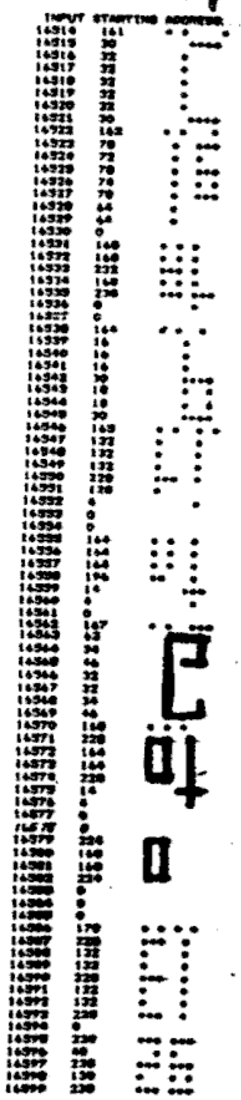


Fig. 4

SPECIAL NOTES

a) TS 2068 owners should already be familiar with the generation of user defined graphics.

b) Use graph paper to lay out your characters.

c) You need not use 8 bytes (in fact our "face" could have been done in 5 bytes, four bytes of data and one attribute "byte"). But, this requires a sound understanding of how the software works.

WHERE TO BUY IT?
(Not Recommendations - just observations)

47th Street Photo		Timex cost
TS 2068	$99.99	95
TS 2040	55.00	49 - 3 Parts

Software (2000) $15 & up

Commack - Consumers Corner - TS/2068

OR Byte, Softsync - Timex Software
Timeworks
ALL TYPES $4.95

New Literature Received:

TUG newsletter - from the Gainesville, Fla. User's Group

Thomas B. Woods PRO/FILE 2068 now available for $29.95 Total Documentation to follow

GETTING THE LED OUT

In any number of monitoring and/or control uses for your ZX/TS, you may find that using a TV set to check on your system is inconvenient or impractical. This would be so, for example, in outdoor or dirty environments or when you simply wish to check the status of one or two system parameters without the bother of hooking up a TV or monitor. One fairly easy and inexpensive way to do this is to use seven-segment LED (light emitting diode) displays as your system output. Your ZX/TS bus doesn't have enough power in reserve to directly drive these displays, but a few simple IC's and an I/O board can give you this capability.

You must have some sort of 8 bit output port. I used Emer-Z's Report Generator board, but JK Audio's 310 and even Byte Back's BB-1 module (without the relays) should be useable as a buffered output from your computer. The only other hardware you'll need, aside from miscellaneous wire and some resistors, is a 7416, hex inverting buffer; a 4511 BCD (Binary Coded Decimal)-to-7 segment decoder driver; and a 3 character, 7 segment LED display (MAN3). A schematic of this simple circuit is given in Fig 1.

This circuit represents a "middle ground" approach to producing a 3 digit output display. More hardware (latches etc.) could have been used, resulting in permanently lit displays, or we could have used software to create a controlling outer loop to keep the monitoring function as the main operating system. The combination of hardware and software I chose represents a system which uses multiplexing to provide the appearance of a full three digit display.

As you can see from Figure one, we are using the ZX's data lines D_0 through D_3 to send out the Binary coded decimal value of one of three digits (units, tens or hundreds). The correct digit is determined by the data lines D_4 through D_6 which complete the path to ground for the appropriate display. Line D_7 enables and latches the last value from lines D_0 to D_3 into the 4511 which in turn, drives the appropriate elements of the display. Multiplexing is provided by the ML software driver routine shown in Fig 2.

After data is POKEd into the three buffer locations (fig 3-a) the ML program outputs the specially constructed character code for each digit in sequence. This operation occurs so rapidly that the eye is fooled into thinking each digit is lit all the time. If you'd like to see what's really happening, you can either increase the dwell time between digits or use the BASIC listing in Fig 4. The BASIC listing requires no machine code but is, of course, slow.

The software is applicable to the Emer-Z Report Generator board as it stands. However, you can change the address of the call at 16561(?) to the address of your own OUTPUT subroutine. The Report Generator uses a PIO and is I/O mapped. For memory mapped sytems, your OUTPUT subroutine would probably consist of simply putting the value in the accumulator, LOADing the designated memory location with the value, and returning to the driver.

I built this circuit on a small ACE (all circuit evaluator) board and obtained its power from the +5 volt regulator on the R.G. board. The values for the current limiting resistors are not critical, but should be as large as possible to conserve power, while still giving adequate brightness. Don't overlook the 10K pullup resistors for the 7416, or forget to tie the unused inputs to ground.

Fig 3b gives a BASIC driver which I use to obtain a temperature reading from the A/D converter on the R.G. board and output that value to the LED display. The code shown would actually be a subroutine called from a main menu. The display is normally blank. By touching, say, "T" on my keyboard, I would be selecting a menu item which calls the Temperature output routine. All this can be done without the need of a TV screen, if I know in advance (and I do) which key to press. The system could be expanded, with more chips, lots of software and perhaps even LCD displays to actually let you program your computer without a TV. A consideration here would be to use hex buffer drivers instead of BCD.

Note too, that I used "junk box" parts from around my shop to build this particular display. A proper design would use perhaps a 74LS248 for the latch driver (pinout is the same as the 4511). Also remember not to output codes which will turn on all three digits at one time (e.g., code 01111000_B produces all 8's), as current drain will be very high.

LIST GROUP

VU-3D OUTPUT - Shaded

MAG=001.83 ROT 347,259 Z=+00270

LEDOUT

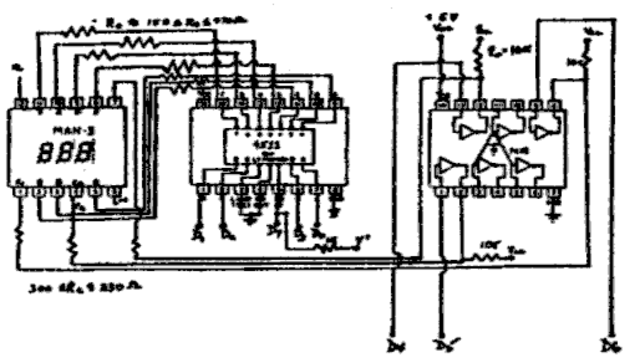


Fig 1 (a)

LEDOUT CODE SENT:

DIGIT	HUNDREDS DIGIT	TENS DIGIT	UNITS DIGIT
0	64	32	16
1	65	33	17
2	66	34	18
3	67	35	19
4	68	36	20
5	69	37	21
6	70	38	22
7	71	39	23
8	72	40	24
9	73	41	25
N	(N+64)	(N+32)	(N+16)

EXAMPLE:

To send the number 888, transmit the following codes in sequence:

24,40,72

i.e.	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
24	0	0	0	1	1	0	0	0
40	0	0	1	0	1	0	0	0
72	0	1	0	0	1	0	0	0

D_3–D_0: BCD CODE; D_6–D_4: DIGIT SELECT; D_7: LATCH ENABLE

Output will actually be:

72 = 8-- which, when
40 = -8- multiplexed, looks
24 = --8 like 888

Fig 1 (b)

LEDOUT

NAME	ADDRESS DEC	HEX	MNEMONIC	COMMENTS	CODE
LEDOUT	16540	409C	LD BC,00FF	Load # of Cycles	01FF00
MAIN	16545	40A1	LD HL,4097		219740
	16548	40A4	LD A,4(5)	# of Digits to send	3E04
ENTRY	16550	40A6	PUSH AF	Save A, Flags	F5
	16551		LD A,(HL)	Get Value to be output from 4097FF	7E
	16552		LD 4082,A	16514 is poked with value	328240
	16555		POPA	Back to original digit count	F1
	16556		INC HL	Point to next digit	23
	16557		DEC A	Only (A-1) digits left	3D
	16558	40AE	JUMP Z Loop	(40BA) have we done all 4(5) digits?	CABA40
	16561	40B1	CALL OUTPUT	(2998) Send digit out	CD9829
	16564		CALL DELAY	(40C0) Leave it lit	CDC040
	16567		JUMP ENTRY	(40A6) Put out next one	C3A640
LOOP	16570	40BA	DEC C	Decrement count	0D
	16571	40BB	RZ	We'll do it FF times then return to BASIC	C8
	16572	40BC	JUMP MAIN	(40A1)	C3A140
	16575	40BF	RETURN	Can't get her	C9
DELAY	16576	40C0	LD DE,0040	A simple time delay	114000
	16579	40C3	DEC E	For how long we leave	1D
	16580	40C4	JZ 40CA	Each digit on	CACA40
	16583	40C7	JUMP 40C3		C3C340
	16586	40CA	LD DE,00AA	This is the "outer"	11AA00
	16589	40CD	DEC E	loop of the	1D
		40CE	RZ	Time delay	C8
		40CF	JUMP 40CD	Keep looping	C3CD40
		40D2	RET	can't get here	C9

DATA

NAME	DEC	HEX			
BUF	16535	4097	HUNDREDS DIGIT	FROM 64 to 73	ALL ARE
	16536	4098	TENS	FROM 32 to 41	Special
	16537	4099	UNITS	FROM 17 to 25	BCD
	16538	409A	BLANKING (255)		
	16539,40,41	409B,C,D	SPARE		

- Fig.2

ADDRESS: 16534 4096H

```
16534   26 48 28 18 00 00 FF 00
16542   01 FF 00 21 97 40 3E 05
16550   F5 7E 32 82 40 F1 23 3D
16558   CA BA 40 CD 98 29 CD C0
16566   40 C3 A6 40 0D C8 C3 A1
16574   40 C9 11 40 00 1D CA CA
16582   40 C3 C3 40 11 AA 00 1D
16590   C8 C3 CD 40 C9
```

Hex Dump of Ledout.
(Number to be output is 888)